Éléments de Programmation
Cours 1 - Introduction

Romain Demangeon

1I001 - Section 11 PEIP

06/09/2016

UPMC SORBONNE UNIVERSITÉS

Étudiants

- PEIP: Informatique au 1er et 3eme semestre.
- Section MIPI 11: 136 étudiants, 5 groupes de TD.

Enseignant

- Romain Demangeon, maître de Conférences au LIP6, équipe APR
- romain.demangeon@lip6.fr
- Bureau 25-26.314 - 0144278825
- Site du Cours:
  http://www-apr.lip6.fr/~demangeon/1I001-16.html
- Site de l'UE:
  http://www.licence.info.upmc.fr/lmd/licence/2016/ue/1I001-2016oct/

Informations pratiques

- Supports: à récupérer lors du premier TD.
  - Plan de cours – saison 01.
  - Poly d'exercices – saison 01.
- Supports: à récupérer plus tard.
  - Carte de référence.
- Groupes de TD:

  11.1 Jeudi apres-midi avec Julie Leloup.
  11.2 Jeudi apres-midi avec Maryse Pelletier.
  11.3 Jeudi matin avec Julie Leloup.
  11.4 Lundi apres-midi avec Dominique Bereziat.
  11.5 Lundi apres-midi avec Christophe Marsala.
- Cours à deux vitesses:
  - Transparents bleus: cours (sanctionnable)
  - Transparents marrons: compléments (non-sanctionnable)

Informatique

Wikipédia

L'informatique est le domaine d'activité scientifique, technique et industriel concernant le traitement automatique de l'information via l'exécution de programmes informatiques par des machines : des systèmes embarqués, des ordinateurs, des robots, des automates, etc.

- L'informatique n'est pas (exactement) la science des ordinateurs.
- Science de l'information et du calcul.
- Nombreux domaines (entre autres):
  - Mathématiques discrètes: algorithmique, modèles de calculs, logique, complexité, concurrence, ...
  - Programmation: langages, typage, sémantique, compilation, ...
  - Architecture: multi-processeurs, puces, calcul flottants, ...
  - Réseaux,
  - Robotique,
  - Intelligence artificielle, ...

Programmation

Wikipédia

Un programme informatique est une séquence d'instructions qui spécifie étape par étape les opérations à effectuer pour obtenir un résultat. Il est exprimé sous une forme qui permet de l'utiliser avec une machine comme un ordinateur pour exécuter les instructions. Un programme est la forme électronique et numérique d'un algorithme exprimé dans un langage de programmation - un vocabulaire et des règles de ponctuation destinées à exprimer des programmes.

- Programmer (écrire des programmes), c'est donner des instructions à une machine en vue de lui faire réaliser un résultat.
  - instructions: analogie de la recette de cuisine,
  - machines: PC, processeur, puce, ...
  - résultats:
    - calculer une expression,
    - construire une image,
    - effectuer un geste physique,
    - mettre en page un texte, ...
- Omniprésence: industrie, économie, physique, média, ...
- Medium entre l'homme et la machine:
  - langage de programmation.

Python

Un cours de programmation en Python, et non un cours de Python.

Wikipedia

Python est un langage de programmation objet, multi-paradigme et multi-plateformes. Il favorise la programmation impérative structurée et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions ; il est ainsi similaire à Perl, Ruby, Scheme, Smalltalk et Tcl.
Le langage Python est placé sous une licence libre proche de la licence BSD2 et fonctionne sur la plupart des plates-formes informatiques, des supercalculateurs aux ordinateurs centraux, de Windows à Unix en passant par GNU/Linux, Mac OS, ou encore Android, iOS, et aussi avec Java ou encore .NET. Il est conçu pour optimiser la productivité des programmeurs en offrant des outils de haut niveau et une syntaxe simple à utiliser.
Il est également apprécié par les pédagogues qui y trouvent un langage où la syntaxe, clairement séparée par des mécanismes de bas niveau, permet une initiation aisée aux concepts de base de la programmation.

- Écrire du "vrai" Python 3.
- Sans utiliser toutes les fonctionnalités de Python.
- Objectif principal:
  - Comprendre des éléments de programmation généraux.
- Conséquence secondaire:
  - Connaître des bases d'un langage en particulier.

I1001 - Éléments de programmation

Concepts fondamentaux

- Résolution de problèmes (ingénieurs).
  - "un probleme, une fonction"
- Démarche:
  - Spécification de problèmes informatiques,
  - Approche algorithmique des solutions,
  - Programmation dans le langage Python

Éléments abordés

- Expressions. (01)
- Fonctions. (02)
- Alternatives. (02)
- Boucles. (03)
- Structures de données: listes (06,07), *n*-uplets (07), chaînes (05), ensembles (09,10), dictionnaires (09,10).

Une Fonction

```
def liste_premiers(n):
    """int -> list[int]
       Hypothese : n >= 0

       Retourne la liste des nombres premiers inferieurs a n."""

    # i_est_premier : bool
    i_est_premier = False  # indicateur de primalite

    # L : list[int]
    L = []   # liste des nombres premiers en resultat

    # i : int (entier courant)
    for i in range(2, n):
        i_est_premier = True

        # j : int (candidat diviseur)
        for j in range(2, i - 1):
            if i % j == 0:
                # i divisible par j, donc i n'est pas premier
                i_est_premier = False

        if i_est_premier:
            L.append(i)

    return L
```

Une Fonction

```
def liste_premiers(n):
    """int -> list[int]
       Hypothese : n >= 0

       Retourne la liste des nombres premiers inferieurs a n."""

    # i_est_premier : bool
    i_est_premier = False  # indicateur de primalite

    # L : list[int]
    L = []   # liste des nombres premiers en resultat

    # i : int (entier courant)
    for i in range(2, n):
        i_est_premier = True

        # j : int (candidat diviseur)
        for j in range(2, i - 1):
            if i % j == 0:
                # i divisible par j, donc i n'est pas premier
                i_est_premier = False

        if i_est_premier:
            L.append(i)

    return L
```

- Définition de fonction: `def liste_premiers(n):`
- Variables: `i_est_premier = False`
- Expressions: `1, i - 1, i % j == 0`
- Applications: `liste_premiers(30)`
- Instructions: alternative `if i_est_premier: ...` (Cours 2)
  boucle `for j in range(2, i - 1): ...` (Cours 3)
- Commentaires, Spécification, Indentations.

Expressions

Définition

Une expression en informatique est une écriture formelle textuelle que l'on peut saisir au clavier.

- Expressions contraintes par un langage (une grammaire).
- Évaluables: valeur obtenue par calcul de la machine.
  - première utilisation d'une machine: évaluer des expressions.
- Deux sortes d'expressions:
  - Expressions atomiques (ou simples):
    - objets informatiques manipulables les plus simples,
    - nombres, chaînes,
    - s'évaluent en elle-même.
  - Expressions composées:
    - formées de sous-expressions (simples ou composées)
    - expressions arithmétiques: sommes, produits, applications de fonctions (max), ...
    - l'évaluer c'est calculer sa valeur (en calculant la valeur des sous-expressions).

Expressions atomiques

- Une expression atomique `v` s'évalue en `v`, sans calcul. (x)
- L'évaluation est un processus complexe:
  1. Lecture de l'expression (utilisation de la grammaire).
  2. Représentation interne de l'expression (codage en objet).
  3. Evaluation de l'expression: résultat (ou non).
  4. Affichage du résultat.

  - un seul `42` "à l'oeil nu", mais plusieurs entités informatiques.

- Une expression possède un type qu'on peut obtenir avec `type`. (x)
- La représentation interne d'un type est une classe d'objets.

Types d'expressions atomiques

Booléens

Expressions logiques, indispensables pour calculer (alternative, boucles).
`True` signifie vrai et `False` signifie faux. (x)
Leur type est `bool`. (x)

Entiers

Écrits en notation mathématique usuelle, de taille arbitraire. (x)
Leur type est `int`. (x)

Types d'expressions atomiques (suite)

Constantes à virgule flottante

Nombres avec virgule (`3.14`) et/ou exposant (`-4.3e-3`). (x)
Approximations informatiques de nombres rationnels.
Branche de l'informatique à part entière.
Les entiers sont convertis en flottants si nécessaire. (x)
On donne le type `Number` à l'ensemble contenant les entiers et les flottants.

Chaînes de caractères

Texte encadré par des apostrophes ou guillemets doubles. (x)
Objets du Cours 5.
Leur type est `str`.

Expressions composées: opérateurs arithmétiques

Définition

Les expressions composées sont formées de combinaisons de sous-expressions, atomiques ou elles-mêmes composées.

- Pour le premier cours/TD/TME: expressions arithmétiques.
  - calculs simples sur les nombres.
- Composition: opérateurs et applications.

Opérateurs arithmétiques:

- opérateurs binaires sur les nombres: addition, soustraction, produit, divisions, ... (x)
- l'opposé, opérateur unaire (x)
- les parenthèses (x)

Expressions composées: opérateurs arithmétiques (suite)

Divisions

Deux types de divisions:

- Division entière (ou division euclidienne) `//` : prend deux entiers, rend un entier. (x)
- Division flottante `/`: prend deux nombres, rend un flottant. (x)

Le reste de la division euclidienne s'obtient avec le modulo `%`. (x)

- En l'absence de parenthèses, la priorité des opérateurs est standard (comme en mathématiques). (x)

Expressions composées: applications de fonctions

- Python dispose de plusieurs fonctions prédéfinies (primitives), accessibles plus ou moins directement.
- Pour appliquer une fonction à une expression, il faut connaître sa spécification, incluant, entre autres, les types de ses entrées et de son résultat et les hypotheses d'application.
  - en mathématiques on écrit par exemple $f : \mathcal{L}(\mathbb{R}^n, \mathbb{R}^m) \to \mathbb{N}$
- En Python, la fonction `sqrt` par exemple est spécifiée par:

```
def sqrt(x):
  """Number -> float
     Hypothese: x >= 0
     retourne la racine carree de x."""
```

- La spécification d'une fonction se compose de:
  1. une en-tête qui donne le nom et ses paramètres,
  2. une signature qui indique le type des paramètres et de la valeur de retour de la fonction.
  3. (si nécessaire) des hypothèses d'application de la fonction.
  4. une description qui indique quel problème résout la fonction.

Expressions composées: applications de fonctions (suite)

Principe d'évaluation

Pour évaluer `fun( arg1, arg2, ..., argn)`:

1. On commence par évaluer `arg1`, puis `arg2`, ..., puis finalement `argn`.
2. On remplace l'appel de fonction (l'application) par sa valeur de retour.

(x)

- Les primitives autorisées en 1I001 sont listées sur la carte de référence.
- les primitives sont classées par Python en bibliothèques appelées modules.
  - pour charger un module on utilise `import`
  - exemple de module: `math` qui contient, entre autres, `math.sin` et `math.pi`. (x)

Evaluation des expressions arithmétiques

Principe général

Pour évaluer une expression arithmétique `e`:

1. Si `e` est atomique, on retourne la valeur `e`.
2. Si `e` est composée, alors:
   - 2.1 On détermine la sous-expression `e1` de `e` à évaluer en premier.
     - On utilise la règle de priorité des opérateurs.
     - On procède de gauche à droite et de haut en bas.
   - 2.2 On évalue `e1`.
   - 2.3 On évalue `e` dans laquelle on remplace `e1` par sa valeur.

(T): Evaluation de `(3 + 5) * ( max(2, 9) - 5 ) - 9`

Définition de Fonctions

- Avec seulement des primitives: calculatrice améliorée.
- Programmation: définition de fonctions par le programmeur.
- Les fonctions ont une place centrale en informatique.
- Elles permettent de paramètrer, d'automatiser et de généraliser des calculs.

Définition de Fonctions: exemple

Problème: Calculer le périmètre d'un rectangle défini par sa longueur et sa largeur.

- formule mathématique: $2 * (L + l)$
- si $l = 2$ et $L = 3$, on saisit `2 * (2 + 3)` (x)
- en mathématiques, on définirait la fonction périmètre par:

$$\begin{array}{rccc} p: & \mathbb{N} \times \mathbb{N} & \rightarrow & \mathbb{N} \\ & (l, L) & \mapsto & 2 * (l + L) \end{array}$$

-
```
def perimetre(largeur, longueur):
    """int * int -> int
    hypothese : (longueur >= 0) and (largeur >= 0)
    hypothese : longueur >= largeur
    retourne le perimetre du rectangle defini par sa largeur et sa longueur."""

    return 2 * (largeur + longueur)
```

- on peut la tester avec `assert perimetre(2, 3) == 10`

UPMC SORBONNE UNIVERSITÉS

Définition de fonction: étapes

Pour définir une fonction, on passe par les étapes suivantes:

1. Spécification du calcul effectué par la fonction:
   1.1 en-tête de la fonction.
   1.2 signature de la fonction.
   1.3 hypothèses pour son application.
2. Implémentation de l'algorithme calculant le résultat.
3. Validation de la fonction par un jeu de test.

```
def perimetre(largeur, longueur):
    """int * int -> int
    hypothese : (longueur >= 0) and (largeur >= 0)
    hypothese : longueur >= largeur
    retourne le perimetre du rectangle defini par sa largeur et sa longueur."""

    return 2 * (largeur + longueur)

# Jeu de tests
assert perimetre(2, 3) == 10
assert perimetre(4, 9) == 26
assert perimetre(0, 0) == 0
assert perimetre(0, 8) == 16
```

Spécification

Définition

La spécification d'une fonction est la partie du code qui décrit le problème que la fonction est censée résoudre.

- rôle: permettre à un programmeur (y compris soi-même) de comprendre comment utiliser la fonction.
- en-tête:
  - introduit par `def`
  - donne le nom de la fonction et ceux de ses paramètres.
- indentation: 4 espaces/une tabulation.
- signature:
  - type des paramètres séparés par $*$
  - type du résultat.
- des hypothèses
  - expressions logiques que doivent vérifier les paramètres.
- une description du calcul effectué par la fonction.

Implémentation

Définition

L'implémentation d'une fonction est l'écriture de l'algorithme qui calcule le résultat de la fonction dans un langage informatique.

- le corps de la fonction est composé d'instructions.
- premier cours: une unique instruction `return expr`
- évaluation de l'instruction `return expr`:
  1. On calcule la valeur de `expr`.
  2. On retourne à l'appelant de la fonction le résultat.
- évaluation de la fonction: Cours 2.
- Plusieurs solutions au problème que la fonction résout: d'autres implémentations.

```
def perimetre(largeur, longueur):
    """int * int -> int
    hypothese : (longueur >= 0) and (largeur >= 0)
    hypothese : longueur >= largeur

    retourne le perimetre du rectangle defini par sa largeur et sa longueur."""
    return (largeur + longueur) + (largeur + longueur)
```

Validation

Approche Contractuelle

- un client avec un problème,
- un programmeur avec une solution.
  - solution = une fonction.

- Problème posé $\leftrightarrow$ Fonction pour le résoudre
- Solution: Définition (Spécification + Implémentation) + Validation
- Validation: montrer que la fonction "marche": calcule bien une solution au problème.
  - problème avec contrat,
  - solution avec tests.
- Appeler la fonction sur des arguments. (x)
- Tester avec des arguments qui respectent la spécification. (x)
- Jeu de tests:
  - expressions d'appel valides.
  - couvrir suffisamment de cas (difficile).

Typage

Donner un type à une expression c'est indiquer la nature d'une expression.

- Objectifs:
  - Vérifier les appels de fonctions.
  - Valider le code (homogénéité).
  - Gérer la mémoire.
- Typage plus ou moins forts
  - OCaml: `float_of_int(x) +.  2.3`
  - Javascript: `(2 + 3) + " saucisses"`
- Typage explicite: le programmeur doit lui-même indiquer les types (déclarations).
- Typage implicite: le type est inféré par un programme (algorithme d'unification).

Sous-Typage

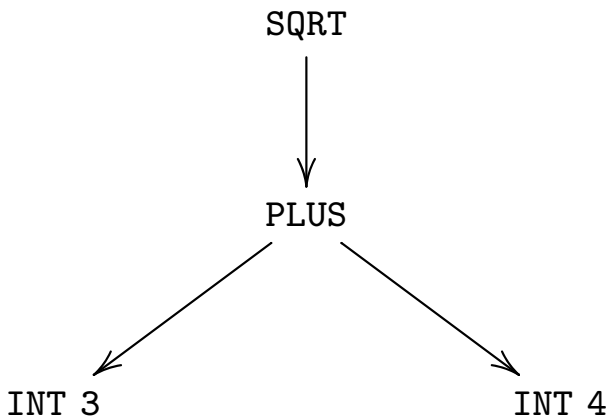
Définition

Un type A est un sous-type de B si toutes les expressions (les objets) de type A sont aussi de type B.

-
  - `int` est un sous-type de `Number`.
  - "entier naturel" est un sous-type de "entier".
  - "poisson" est un sous-type de "animal".
- Si on a besoin d'une expression de type B, et que A est un sous-type de B, on peut prendre une expression de type A.
  - si f prend un entier, je peux calculer $f(3)$.
  - si j'ai besoin d'un animal, je peux prendre un poisson.
- Attention au sens:
  - si f prend un entier naturel, je ne peux pas (forcément) calculer $f(-3)$.
  - si j'ai besoin d'un poisson, je ne peux pas (forcément) prendre un serpent.
- Dans les signatures des fonctions: + général pour les paramètres, + particulier pour le résultat.
- Héritage dans les langages objets (11).

Grammaires d'expressions

- Compilation: domaine de l'informatique qui s'intéresse à la traduction d'un langage dans un autre.
- Fondement de la programmation: traduction d'un langage "compréhensible" (Python) en langage machine.
  - point de détail: Python est interprété et non compilé.
- Analyse lexicale: séparation du code en jetons.
  - `math.sqrt(3 + 4)` $\longrightarrow$ reconnaitre `sqrt`, 3, 4, l'opérateur +, les parenthèses.
- Analyse syntaxique: organisation des jetons en arbre syntaxique.
  -



Grammaires d'expressions (II)

- Les Grammaires permettent d'exprimer le code reconnaissable par le compilateur/l'interprêteur.
- Définition à l'aide de "graînes" `S ::= E1 | E2 | ... | EN`
  - formellement, point fixe d'une fonction (théorème de Knaster-Tarski).
- Grammaire des entiers `N ::= O | Succ(N)`
- Grammaire de l'arithmétique `N ::= O | Succ(N) | Plus(N,N) | Sous(N,N) | Mult(N,N)`
- Grammaire de la Carte de référence.

Conclusion

Résumé

- Appliquer les principes d'évaluation pour les expressions arithmétiques.
- Définir des fonctions simples consistant à paramétrer des expressions arithmétiques, avec :
  - une spécification précise du problème résolu (en-tête, signature et description),
  - une implémentation du corps de la fonction,
  - un jeu de tests complet permettant de valider la fonction.

Semaine prochaine (13/09)

Ces Fonctions qui nous gouvernent.